In [ ]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:

```
%cd gdrive/MyDrive/probstat
```

```
/content/gdrive/MyDrive/probstat
```

In [ ]:

```
import pandas as pd
import numpy as np
```

In [ ]:

```
def get_daily_counts(x):
  y=np.array([0])
  y=np.append(y,x)
  y=y[:-1]
  return x-y
```

In [ ]:

```
def outliers_and_data_cleaning(x):
  Q1 = np.percentile(x, 25, interpolation = 'midpoint')
  Q3 = np.percentile(x, 75, interpolation = 'midpoint')
  #tukey's rule outliers are with first_quartile-1.5IQR and third_quartile+1.5IQR
  IQR=Q3-Q1
  left=Q1-1.5*IQR
  right=Q3+1.5*IQR
  mean=np.mean(x)
  print("Lower bound",left)
  print("Upper bound",right)
  #replace with mean if the outliers are found
  for i in range(x.size):
    if(x[i]<left and x[i]!=0):
      x[i]=mean
    elif(x[i]>right and x[i]!=0):
      x[i]=mean
  return x
```

In the above function, we apply tukey's rule to identify the left and right bounds for the data

In [ ]:

```
def remove_outliers_and_create_new_file():
    dataframe = pd.read_csv('../15.csv')
    ne_confirmed = dataframe['NE confirmed'].to_numpy()
    nd_confirmed = dataframe['ND confirmed'].to_numpy()
    ne_deaths = dataframe['NE deaths'].to_numpy()
    nd_deaths = dataframe['ND deaths'].to_numpy()

    #get the daily counts from each state and each category
    ne_confirmed_per_day = get_daily_counts(ne_confirmed)
    nd_confirmed_per_day = get_daily_counts(nd_confirmed)
    ne_deaths_per_day = get_daily_counts(ne_deaths)
    nd_deaths_per_day = get_daily_counts(nd_deaths)

    #remove outliers by replacing them with mean and not changing zeros
    print("nd_confirmed_per_day")
    nd_confirmed_per_day = outliers_and_data_cleaning(nd_confirmed_per_day)
    print("ne_confirmed_per_day")
    ne_confirmed_per_day = outliers_and_data_cleaning(ne_confirmed_per_day)
    print("nd_deaths_per_day")
    nd_deaths_per_day = outliers_and_data_cleaning(nd_deaths_per_day)
    print("ne_deaths_per_day")
    ne_deaths_per_day = outliers_and_data_cleaning(ne_deaths_per_day)

    #append to a new file
    dataframe['ne_confirmed_per_day'] = ne_confirmed_per_day
    dataframe['nd_confirmed_per_day'] = nd_confirmed_per_day
    dataframe['ne_deaths_per_day'] = ne_deaths_per_day
    dataframe['nd_deaths_per_day'] = nd_deaths_per_day
    dataframe.to_csv('../15_updated.csv')
```

In the above the function we take each column data, find the outliers, replace with the mean of rhe data and output to a new file

In [ ]:

```
remove_outliers_and_create_new_file()
```

```
nd_confirmed_per_day
Lower bound -333.75
Upper bound 592.25
ne_confirmed_per_day
Lower bound -536.75
Upper bound 1161.25
nd_deaths_per_day
Lower bound -4.5
Upper bound 7.5
ne_deaths_per_day
Lower bound -7.5
Upper bound 12.5
```

Approach

- We used tukey's rule to find the lower and upper bounds where outliers exists

- Since removing all the outliers may cause gaps in the time series data, we replace the outliers with the mean data
- changing the outliers does not cause any changes to the data distribution.

In [ ]:

```

```

In [12]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, ca
ll drive.mount("/content/gdrive", force_remount=True).
```

In [13]:

```
%cd gdrive/MyDrive/probstat
```

```
[Errno 2] No such file or directory: 'gdrive/MyDrive/probstat'
/content/gdrive/MyDrive/probstat
```

In [ ]:

```
%pwd
```

Out[22]:

```
'/content/gdrive/My Drive/probstat'
```

In [14]:

```
import pandas as pd
import numpy as np
```

In [25]:

```
def get_beta(train_x, train_y):
    train_x_transpose = np.transpose(train_x)
    #find the transpose of X
    x_x_t = np.matmul(train_x_transpose, train_x)
    #We know B= x^-1*X*Y
    beta = np.matmul(np.matmul(np.linalg.inv(x_x_t), train_x_transpose), train_y)
    return beta
```

The above function is used for finding the beta values from the x and y values

In [ ]:

```
def pad_one(x):
    y=np.array([1])
    y=np.append(y,x)
    return y
```

The above function is used to append 1 to the begining of an array for the constant beta term

In [17]:

```python
def get_regression_data(x, p):
    n = x.size
    #create a matrix with all zeros
    x_data_without_ones = np.ones((n - p, p))
    x_data = np.ones((n - p, p+1))
    y_data = np.ones(n - p)
    #get p values for each row
    for i in range(p, n):
        y_data[i - p] = x[i]
        #pad zeros for constant beta term
        x_data_without_ones[i - p, :] = np.reshape(x[i - p:i], (1, p))
        x_data[i - p, :] = pad_one(x_data_without_ones[i - p, :])
    return x_data, y_data
```

The above function is used to get the time series data for the past p days and the predicted value on the p+1 day

In [18]:

```python
def AR(x, p):
    print("AR("+str(p)+")")
    x_three = x[70:91]
    x_four = x[91:100]
    n = x_four.size
    mape = 0
    mse = 0
    #change the input data based on the value of p.i.e, if p=3 then three consucitive recor
    x_data, y_data = get_regression_data(x[70:100], p)
    for i in range(n - 1):
        #Beta values are calculated by appending 1 to each row and doing the matrix inverse
        #beta is calculated at each stage
        beta = get_beta(x_data[0:21 + i - p, :], y_data[0:21 - p + i])
        #find the predicted array
        pred = beta * x_data[21 + i - p + 1, :]
        #find the predicted value
        pred = np.sum(pred)
        # if the denominator is zeo in the MAPE calculation changed the denominator to 1
        if (x_four[i] == 0):
            mape = mape + abs((x_four[i] - pred))
        else:
            mape = mape + abs((x_four[i] - pred) / x_four[i])
        mse = mse + ((x_four[i] - pred) * (x_four[i] - pred))
    mape = 100 / n * (mape)
    mse = mse / n
    print("Mean Absolute Percentage Error", mape)
    print("Mean Squared error", mse)
    return mape, mse
```

- The above function is used to find the predicted value using auto regression method.
- The metrics are calculated after the each prediction and the final metrics are printed

In [19]:

```
def get_predicted_value(x, alpha):
    #x is the data of the last p-1 days
    n = x.size
    rev_alpha = 1 - alpha
    coeff = 1
    pred = 0
    # when we expand the EWMA we get the below coefficients for each term
    for i in range(n):
        pred = pred + (x[n - i - 1] * (coeff))
        coeff = coeff * rev_alpha
    pred = pred * alpha
    #return the predicted value on the pth day
    return pred
```

Get the pth day predicted value based on the alpha and previous p-1 days data

In [20]:

```
def EWMA(x, alpha):
    print("EWMA("+str(alpha)+")")
    #x_three is the data of the first weeks
    x_three = x[70:91]
    #x_four is the data of the fourth week
    x_four = x[91:100]
    #number of predicted days in the fourth week
    n=x_four.size
    mse = 0
    mape = 0
    for i in range(n):
        #get the predicted value on t-day from t-1 days data
        #t-1 days data is passed at each iteration
        pred = get_predicted_value(x[70:91 + i], alpha)
        #if the denominator is zeo in the MAPE calculation changed the denominator to 1
        if(x_four[i]==0):
            mape = mape + abs((x_four[i] - pred))
        else:
            mape = mape + abs((x_four[i] - pred) / x_four[i])
        mse = mse + ((x_four[i] - pred) * (x_four[i] - pred))
    mape = (100 / n) * (mape)
    mse = mse / n
    print("Mean Absolute Percentage Error", mape)
    print("Mean Squared error", mse)
    return mape, mse
```

Perform EWMA on the data to find the predicted value of the fourth week based on the data of the data before the current day

In [21]:

```
def run_all_models(x):
    #Autoregressive model with p
    AR(x, 3)
    AR(x, 5)

    #Exponential weighted moving average with alpha
    EWMA(x, 0.5)
    EWMA(x, 0.8)
```

Run all the models with the parameters for an attribute

In [22]:

```
def q1_a():
    #get the data from the file
    dataframe = pd.read_csv('../15_updated.csv')
    ne_confirmed = dataframe['ne_confirmed_per_day'].to_numpy()
    nd_confirmed = dataframe['nd_confirmed_per_day'].to_numpy()
    ne_deaths = dataframe['ne_deaths_per_day'].to_numpy()
    nd_deaths = dataframe['nd_deaths_per_day'].to_numpy()


    print("NE Confirmed")
    run_all_models(ne_confirmed)
    print("ND Confirmed")
    run_all_models(nd_confirmed)
    print("NE Deaths")
    run_all_models(ne_deaths)
    print("ND Deaths")
    run_all_models(nd_deaths)
```

In [24]:

```
q1_a()
```

```
NE Confirmed
AR(3)
Mean Absolute Percentage Error 32.96850430025964
Mean Squared error 8349.724692558942
AR(5)
Mean Absolute Percentage Error 52.11422476794114
Mean Squared error 18891.37117302536
EWMA(0.5)
Mean Absolute Percentage Error 36.87492078436932
Mean Squared error 12446.65933430085
EWMA(0.8)
Mean Absolute Percentage Error 32.35788333457912
Mean Squared error 10892.235627411601
ND Confirmed
AR(3)
Mean Absolute Percentage Error 404.9411215660536
Mean Squared error 155572.02981181236
AR(5)
Mean Absolute Percentage Error 938.3191264733968
Mean Squared error 340283.1046553315
EWMA(0.5)
Mean Absolute Percentage Error 39.24418879275243
Mean Squared error 331.4057628966122
EWMA(0.8)
Mean Absolute Percentage Error 28.747277708178856
Mean Squared error 187.76483880995954
NE Deaths
AR(3)
Mean Absolute Percentage Error 47.09115008203833
Mean Squared error 1.7882953488439088
AR(5)
Mean Absolute Percentage Error 58.42903933084258
Mean Squared error 1.8807861237175874
EWMA(0.5)
Mean Absolute Percentage Error 11.121908575296402
Mean Squared error 1.7777744125300556
EWMA(0.8)
Mean Absolute Percentage Error 17.77777785059521
Mean Squared error 1.9377777777777219
ND Deaths
AR(3)
Mean Absolute Percentage Error 54.33218017569266
Mean Squared error 0.49843168790177056
AR(5)
Mean Absolute Percentage Error 63.589199494864296
Mean Squared error 0.8226222970787114
EWMA(0.5)
Mean Absolute Percentage Error 55.5695202615526
Mean Squared error 0.5855089373064304
EWMA(0.8)
Mean Absolute Percentage Error 58.56516928220713
Mean Squared error 0.6996864856289055
```

In [22]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount,
call drive.mount("/content/gdrive", force_remount=True).
```

In [18]:
```
cd gdrive/MyDrive/
```

```
/content/gdrive/MyDrive
```

In [23]:
```
import pandas as pd
import numpy as np
import math
```

```
In [24]: df = pd.read_csv("15_daily_filtered_mean.csv")
         df_feb = df[(df['Date'] >= '2021-02-01') & (df['Date'] <= '2021-02-28')]
         df_mar = df[(df['Date'] >= '2021-03-01') & (df['Date'] <= '2021-03-31')]
         df_feb = df_feb.loc[:, 'ND confirmed':]
         df_mar = df_mar.loc[:, 'ND confirmed':]
         print(df_mar)
```

```
     ND confirmed  NE confirmed  ND deaths  NE deaths
404           108           377          2          2
405           113           276          1          7
406           110           358          0          1
407            94           325          0         20
408            82           337          1          1
409            33           386          0          0
410            29           141          0          0
411            97           195          0          2
412            99           301          3         10
413           108           281          3          2
414           120           265          1          2
415            93           287          2          3
416            34           165          0          0
417            25           114          0          0
418           148           317          1          1
419           133           216          1          1
420           119           276          0          1
421           113           254          1          4
422             0           395          0          2
423             0           137          0          0
424             0           179          0          0
425             0           245          1          2
426           486           206          3         26
427           197           286          1          9
428           183           363          1          1
429           159           531          0          0
430            47           225          0          0
431            55           126          0          0
432           245           350          0          2
433           216           408          0          3
434           205          2319          0          2
```

One Sample W-Test

In [27]:

```
feb_mean = df_feb.mean()
cols = len(df_feb.columns)
n = len(df_mar)

lambda_MLE = df_mar.sum().div(n)
se_MLE = df_mar.var().div(math.sqrt(n))

w = (lambda_MLE - feb_mean).div(se_MLE)

print(w)
# alpha=0.05
reject_hypo = (w <= -1.64)
print("1-Sample W-Test")
print("H0: mean(mar)>=mean(feb), Reject:\n ", reject_hypo)
print("######################################################################
```

```
ND confirmed    0.020000
NE confirmed   -0.000110
ND deaths      -0.867704
NE deaths      -0.411946
dtype: float64
1-Sample W-Test
H0: mean(mar)>=mean(feb), Reject:
  ND confirmed    False
NE confirmed    False
ND deaths       False
NE deaths       False
dtype: bool
########################################################################
```

Results of Walds 1 sample testing for mean of cases and death

Null hypothesis (H0):

the mean of daily cases and the mean of daily deaths for Feb’21 is different from the corresponding mean of daily values for March’21

Mean of Feb’21 cases/deaths **<=** Mean of March’21 cases/deaths.

Alternate hypothesis(H1):

Mean of Feb’21 cases/deaths **>** mean of March’21 cases/deaths.

Procedure :

We have taken the guess value as mean of daily values from Feb’21 cases/deaths and alpha = 0.05 as given in documentation and the MLE estimator for mean of daily values from Mar’21 cases/deaths becomes the sample mean of daily values from Mar’21 cases/deaths.

Result:

walds 1 sample testing for mean of state ND confirmed cases is w=0.020000

walds 1 sample testing for mean of state NE confirmed cases is w=-0.000110

walds 1 sample testing for mean of state ND deaths is w=-0.867704

walds 1 sample testing for mean of state ND deaths is w=-0.411946

which are greater than z_alpha = -1.64 so reject the NULL hypothesis

Type *Markdown* and LaTeX: α^2

One sample Z-Test

In [ ]:
```
# True Variance
df_t = df.loc[:, 'ND confirmed':]
mean = df.loc[:, 'ND confirmed':].mean()

diff_sum = df_t.sub(mean).sum()
True_Sample_Dev = diff_sum.mul(diff_sum).div(len(df_t) - 1) ** (1 / 2)
#
print("1-Sample Z-Test")
z = (df_mar.mean() - feb_mean).div(True_Sample_Dev.div(math.sqrt(n))).abs()
print(z)
reject_hypo = (z <= -1.64)
print("H0: mean(mar)>=mean(feb), Reject:\n ", reject_hypo)
print("####################################################################
```

```
1-Sample Z-Test
ND confirmed    2.334509e+13
NE confirmed    1.015164e+12
ND deaths       7.242974e+12
NE deaths       4.125351e+13
dtype: float64
H0: mean(mar)>=mean(feb), Reject:
  ND confirmed    False
NE confirmed    False
ND deaths       False
NE deaths       False
dtype: bool
########################################################################
```

Result of Z testing for mean of cases and death

Null hypothesis (H0):

Mean of Feb’21 cases/deaths **<=** Mean of March’21 cases/deaths.

Alternate hypothesis(H1):

Mean of Feb’21 cases/deaths **>** mean of March’21 cases/deaths.

Result:

Z-sample testing for mean of state ND confirmed cases is w=2.334509e+13

Z-sample testing for mean of state NE confirmed cases is w=1.015164e+12

Z-sample testing for mean of state ND deaths is w=7.242974e+12

Z-sample testing for mean of state ND deaths is w=4.125351e+13

which are greater than z_alpha = -1.64 so reject the NULL hypothesis

One Sample T-test

In [ ]:
```
print("1-Sample T-test")

df_t = df_mar.loc[:, 'ND confirmed':]
mean = df_mar.loc[:, 'ND confirmed':].mean()

diff_sum = df_t.sub(mean).sum()
Sample_Dev = diff_sum.mul(diff_sum).div(len(df_t) - 1) ** (1 / 2)

t = (df_mar.mean() - feb_mean).div(Sample_Dev.div(math.sqrt(n))).abs()
print(t)

reject_hypo = (t <= -1.6973)
print("H0: mean(mar)>=mean(feb), Reject:\n ", reject_hypo)
print("####################################################################
## Comment about applicability of tests
```

```
1-Sample T-test
ND confirmed    3.594604e+16
NE confirmed    3.821252e+14
ND deaths       9.644333e+14
NE deaths       6.224550e+15
dtype: float64
H0: mean(mar)>=mean(feb), Reject:
  ND confirmed    False
NE confirmed    False
ND deaths       False
NE deaths       False
dtype: bool
#######################################################################
```

Result of T 1 sample testing for mean of cases and death

Null hypothesis (H0):

Mean of Feb'21 cases/deaths <= Mean of March'21 cases/deaths.

Alternate hypothesis(H1):

Mean of Feb'21 cases/deaths > Mean of March'21 cases/deaths.

Procedure :

We have taken the alpha = 0.05,n = 30 as we took 30 days of data as given in documentation and calculated the numerator and denominator of t.

Result:

As the calculated values for mean of state

ND confirmed cases = 3.594604e+16

NE confirmed cases = 3.821252e+14

ND deaths = 9.644333e+14

NE deaths = 6.224550e+15

which are greater than t value -1.6973 we are rejecting the NULL hypothesis.

Two Sample Walds Test

In [ ]:

```
print("2-Sample Walds-test")
D_mean = df_feb.mean() - df_mar.mean()

Var_feb = ((df_feb - df_feb.mean()) ** 2).sum().div(len(df_feb))
Var_mar = ((df_mar - df_mar.mean()) ** 2).sum().div(len(df_mar))
std_dev = (Var_feb / len(df_feb) + Var_mar / len(df_mar)) ** 0.5

w = (D_mean / std_dev).abs()
print(w)
reject_hypo = (w >= 1.96)

print("H0: mean(mar)=mean(feb), Reject:\n", reject_hypo)
print("######################################################################
```

```
2-Sample Walds-test
ND confirmed    1.591266
NE confirmed    0.036141
ND deaths       0.521922
NE deaths       1.723450
dtype: float64
H0: mean(mar)=mean(feb), Reject:
 ND confirmed    False
NE confirmed    False
ND deaths       False
NE deaths       False
dtype: bool
########################################################################
```

Result of Walds 2 sample testing for mean of cases and death

Null hypothesis (H0):

Mean of Feb’21 cases/deaths **=** Mean of March’21 cases/deaths.

Alternate hypothesis(H1):

Mean of Feb’21 cases/deaths **not equal to** Mean of March’21 cases/deaths.

Procedure :

We have taken the alpha = 0.05 as given in documentation and calculated the numerator and denominator of w.

Result:

As the w value for mean of state

ND confirmed cases 1.591266

NE confirmed cases 0.036141

ND deaths 0.521922

NE deaths 1.723450

which are less than 1.96 we are rejecting the NULL hypothesis.

Type *Markdown* and LaTeX: α^2

Two Sample Unpaired T-test

In [ ]:
```
print("2-Sample T-test")

Var_feb = ((df_feb - df_feb.mean()) ** 2).sum().div(len(df_feb) - 1)
Var_mar = ((df_mar - df_mar.mean()) ** 2).sum().div(len(df_mar) - 1)

pool_std = (Var_feb / len(df_feb) + Var_mar / len(df_mar)) ** 0.5

t = ((df_feb.mean() - df_mar.mean()) / pool_std).abs()
print(t)

reject_hypo = (t >= 2.0423)

print("H0: mean(mar)=mean(feb), Reject:\n", reject_hypo)
print("####################################################################
```

```
2-Sample T-test
ND confirmed    1.564428
NE confirmed    0.035535
ND deaths       0.512856
NE deaths       1.693888
dtype: float64
H0: mean(mar)=mean(feb), Reject:
 ND confirmed    False
NE confirmed    False
ND deaths       False
NE deaths       False
dtype: bool
#######################################################################
```

Result of T 2 sample unpaired testing for mean of cases and death

Null hypothesis (H0):

Mean of Feb’21 cases/deaths = Mean of March’21 cases/deaths.

Alternate hypothesis(H1):

Mean of Feb’21 cases/deaths **not equal to** Mean of March’21 cases/deaths.

Procedure :

We have taken the alpha = 0.05 n=27, m=30 as given in documentation and calculated the numerator and denominator of t value.

Result:

As the t value for mean of state

ND confirmed cases 1.564428

NE confirmed cases 0.035535

ND deaths 0.512856

NE deaths 1.693888

which are less than 2.0423 we are rejecting the NULL hypothesis.

Applicability Of Tests

Wald's Test

In Wald's we need to have a Asymptomatically Normal estimator. Since N is fairly low, we cannot assume CLT applies in case of sample mean and hence this test is not applicable.

The above reasoning works well for 2 sample test as well, since we need both estimators to be Asymptomatically Normal. Hence this test is also not applicable.

Z-test

In Z-test, we need to have true standard deviation and either the N is large or the X is normally distributed. Since neither of the second requirement is true, we say the test is not applicable.

T-test

In one sample T-test, the requirement is that the data is normally distributed, since this is not the case and N is low as well. We say that the test is not applicable

In unpaired T-test, the required distributions should be independent and need to be normally distributed. In the current scenario, neither is the case so we say that the test is not applicable

In paired T-test, the difference distribution should be normally distributed but we cannot assume the same, hence we say that the test is not applicable

In [1]:
```
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv("/Users/kodalialekhya/Downloads/15_new.csv")
```

In [4]:
```
df = df.drop(columns='Unnamed: 0')
```

In [5]:
```
df.head()
```

Out[5]:

	Date	ND confirmed	NE confirmed	ND deaths	NE deaths	ne_confirmed_per_day	nd_confirmed_per_day
0	2020-01-22	0	0	0	0	0	0
1	2020-01-23	0	0	0	0	0	0
2	2020-01-24	0	0	0	0	0	0
3	2020-01-25	0	0	0	0	0	0
4	2020-01-26	0	0	0	0	0	0

In [6]:
```
data = df[(df.Date>'2020-09-30') & (df.Date<'2021-01-01')]
data.head()
```

Out[6]:

	Date	ND confirmed	NE confirmed	ND deaths	NE deaths	ne_confirmed_per_day	nd_confirmed_per_da
253	2020-10-01	22218	45870	256	494	594	37
254	2020-10-02	22694	46647	264	494	777	47
255	2020-10-03	23134	47070	271	498	423	44
256	2020-10-04	23550	47475	274	500	405	41
257	2020-10-05	23862	47910	277	503	435	31

In [11]:
```
def get_xy(x):

  n = len(x)
  x = sorted(x)
  x_cdf = []
  y_cdf = []
  y_curr = 0

  x_cdf.append(0)
  y_cdf.append(0)

  for i in x:
    y_curr += 1/n
    y_cdf.append(y_curr)
    x_cdf.append(i)

  return x_cdf,y_cdf
```

In [13]:
```
def draw_ecdf(x1, y1, x2, y2, max_diff, max_ind):
    plt.figure(figsize=(20,10))
    plt.step(x1, y1, where="post", label="CDF-D1")
    plt.step(x2, y2, where="post", label="CDF-D2")
    plt.yticks(np.arange(0, 1.1, 1/10))
    plt.title("Empirical CDF")
    plt.xlabel("Sample Points")
    plt.ylabel("Pr[X<x]")
    plt.scatter([max_ind],[0], color='red', marker='x', s=100, label=f
    plt.grid(which="both")
    plt.legend()
    plt.show()
```

In [35]:
```
def ks_1_sample_test(data1,data2, statement, threshold=0.05):
  x1, y1 = get_xy(data1)

  n = len(data2)

  diff=[]
  for i in range(n):
    diff.append( np.absolute( y1[i] - data2[i]  ) )

  max_diff = np.max(diff)

  max_ind = np.argmax(diff)

  if max_diff > threshold:
    print(f"Max value = {max_diff} > C: {threshold}, We reject H0: "+s
  else:
    print(f"Max value = {max_diff} <= C: {threshold}, We reject H0: "+
```

In [36]:
```
mean_nd_confirmed = np.mean(data['nd_confirmed_per_day'].values)
mean_nd_deaths = np.mean(data['nd_deaths_per_day'].values)
```

In [37]:
```
def calc_poisson(param, x):
  return stats.poisson.cdf(x, param)
```

In [38]:
```
x1, y1 = get_xy(data['ne_confirmed_per_day'].values)
val = calc_poisson(mean_nd_confirmed, x1)
ks_1_sample_test(data['ne_confirmed_per_day'].values, val, "The cases
```

```
Max value = 0.9130434782608696 > C: 0.05, We reject H0: The cases fol
low poisson distrubtion
```

Result of 1 sample KS test for last 3 months of 2020 for NE state cases with poisson distribution

Null Hypothesis (H0):

Distribution of last 3 months of 2020 for NE state cases equals poisson distribution

Alternate Hypothesis (H1):

Distribution of last 3 months of 2020 for NE state cases not equals poisson distribution

Procedure:

We have obtained parameters for poisson distribution by using MME on last 3 months for ND state data. We have taken the c = 0.05 and n=92 as given in documentation and calculated the maximum difference of the CDF of the distributions at all the points.

Result:

As the KS test value is 0.913 which is greater than 0.05 we are rejecting the NULL hypothesis.

In [18]:
```
x1, y1 = get_xy(data['ne_deaths_per_day'].values)
val = calc_poisson(mean_nd_deaths, x1)
ks_1_sample_test(data['ne_deaths_per_day'].values, val, "The deaths fo
```

```
Max value = 0.4979724320106867 > C: 0.05, We reject H0: The deaths fo
llow poisson distrubtion
```

Result of 1 sample KS test for last 3 months of 2020 for NE state deaths with poisson distribution

Null Hypothesis (H0):

Distribution of last 3 months of 2020 for NE state deaths equals poisson distribution

Alternate Hypothesis (H1):

Distribution of last 3 months of 2020 for NE state deaths not equals poisson distribution

Procedure:

We have obtained parameters for poisson distribution by using MME on last 3 months for ND state data. We have taken the c = 0.05 and n=92 as given in documentation and calculated the maximum difference of the CDF of the distributions at all the points.

Result:

As the KS test value is 0.497 which is greater than 0.05 we are rejecting the NULL hypothesis.

In [40]:
```
def calc_geometric(param, x):
  return stats.geom.cdf(x, param)
```

In [44]:
```
x1, y1 = get_xy(data['ne_confirmed_per_day'].values)
val = calc_poisson(1/mean_nd_confirmed, x1)
ks_1_sample_test(data['ne_confirmed_per_day'].values, val, "The cases
```

```
Max value = 0.9961051655953749 > C: 0.05, We reject H0: The cases fol
low geometric distrubtion
```

Result of 1 sample KS test for last 3 months of 2020 for NE state cases with geometric distribution

Null Hypothesis (H0):

Distribution of last 3 months of 2020 for NE state cases equals geometric distribution

Alternate Hypothesis (H1):

Distribution of last 3 months of 2020 for NE state cases not equals geometric distribution

Procedure:

We have obtained parameters for geometric distribution by using MME on last 3 months for ND state data. We have taken the c = 0.05 and n=92 as given in documentation and calculated the maximum difference of the CDF of the distributions at all the points.

Result:

As the KS test value is 0.996 which is greater than 0.05 we are rejecting the NULL hypothesis.

In [45]:
```
x1, y1 = get_xy(data['ne_deaths_per_day'].values)
val = calc_poisson(1/mean_nd_deaths, x1)
ks_1_sample_test(data['ne_deaths_per_day'].values, val, "The cases fol
```

```
Max value = 0.7725453890382081 > C: 0.05, We reject H0: The cases fol
low geometric distrubtion
```

Result of 1 sample KS test for last 3 months of 2020 for NE state deaths with geometric distribution

Null Hypothesis (H0):

Distribution of last 3 months of 2020 for NE state deaths equals geometric distribution

Alternate Hypothesis (H1):

Distribution of last 3 months of 2020 for NE state deaths not equals geometric distribution

Procedure:

We have obtained parameters for geometric distribution by using MME on last 3 months for ND state data. We have taken the c = 0.05 and n=92 as given in documentation and calculated the maximum difference of the CDF of the distributions at all the points.

Result:

As the KS test value is 0.772 which is greater than 0.05 we are rejecting the NULL hypothesis.

In [46]:
```
def calc_binomial(n, p, x):
  return stats.binom.cdf(x, n, p)
```

In [23]:
```
var_nd_confirmed = np.var(data['nd_confirmed_per_day'].values)
n = (mean_nd_confirmed * mean_nd_confirmed)/(mean_nd_confirmed - var_r
p = mean_nd_confirmed/n

x1, y1 = get_xy(data['ne_confirmed_per_day'].values)
val = calc_binomial(n, p, x1)
ks_1_sample_test(data['ne_confirmed_per_day'].values, val, "The cases
```

```
Max value = 1.0 > C: 0.05, We reject H0: The cases follow binomial di
strubtion
```

Result of 1 sample KS test for last 3 months of 2020 for NE state cases with binomial distribution

Null Hypothesis (H0):

Distribution of last 3 months of 2020 for NE state cases equals binomial distribution

Alternate Hypothesis (H1):

Distribution of last 3 months of 2020 for NE state cases not equals binomial distribution

Procedure:

We have obtained parameters for binomial distribution by using MME on last 3 months for ND state data. We have taken the c = 0.05 and n=92 as given in documentation and calculated the maximum difference of the CDF of the distributions at all the points.

Result:

As the KS test value is 1 which is greater than 0.05 we are rejecting the NULL hypothesis.

In [24]:
```
var_nd_deaths = np.var(data['nd_deaths_per_day'].values)
n = (mean_nd_deaths * mean_nd_deaths)/(mean_nd_deaths - var_nd_deaths)
p = mean_nd_deaths/n

x1, y1 = get_xy(data['ne_deaths_per_day'].values)
val = calc_binomial(n, p, x1)
ks_1_sample_test(data['ne_deaths_per_day'].values, val, "The cases fol
```

```
Max value = 1.0 > C: 0.05, We reject H0: The cases follow binomial di
strubtion
```

Result of 1 sample KS test for last 3 months of 2020 for NE state deaths with binomial distribution

Null Hypothesis (H0):

Distribution of last 3 months of 2020 for NE state deaths equals binomial distribution

Alternate Hypothesis (H1):

Distribution of last 3 months of 2020 for NE state deaths not equals binomial distribution

Procedure:

We have obtained parameters for binomial distribution by using MME on last 3 months for ND state data. We have taken the c = 0.05 and n=92 as given in documentation and calculated the maximum difference of the CDF of the distributions at all the points.

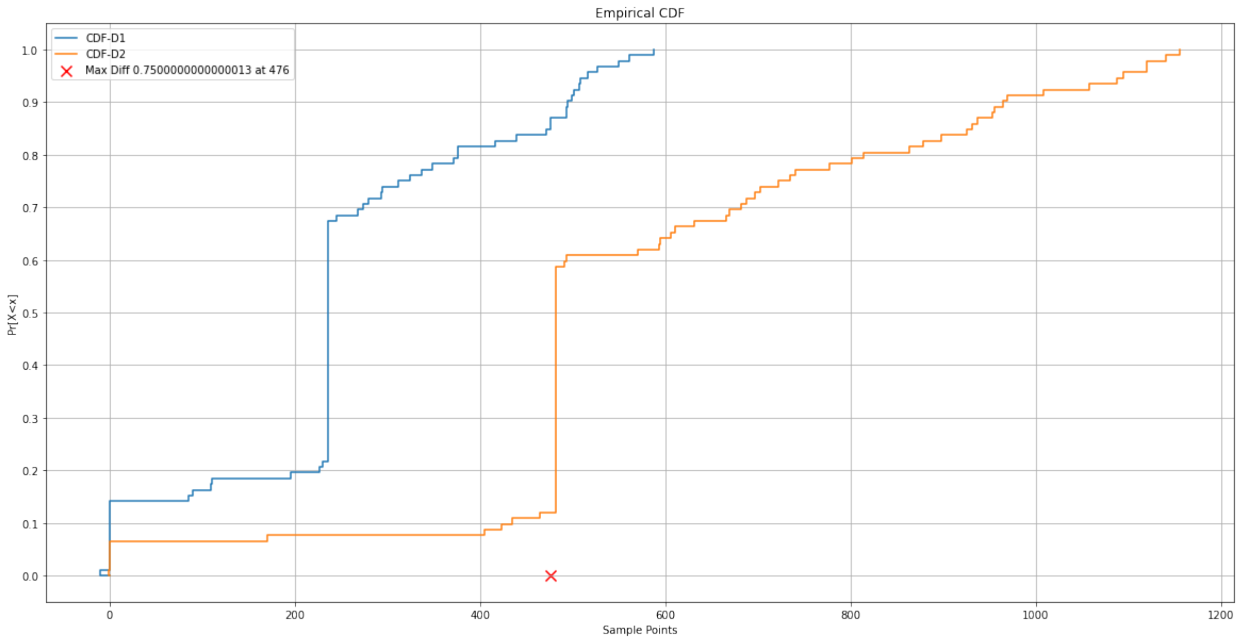
Result:

As the KS test value is 1 which is greater than 0.05 we are rejecting the NULL hypothesis.

In [62]:
```
def ks_2_sample_test(data1,data2, threshold=0.05, draw=True):
  x1, y1 = get_xy(data1)
  x2, y2 = get_xy(data2)

  n = int(min([max(x1),max(x2)])) +10

  y1_all = []
  temp=0
  for i in np.arange(n):
    ind = np.where(np.array(x1) == i)[0]
    if len(ind)==0:
      y1_all.append(temp)
    else:
      y1_all.append(y1[ind[-1]])
      temp = y1[ind[-1]]

  y2_all = []
  temp=0
  for i in np.arange(n):
    ind = np.where(np.array(x2) == i)[0]
    if len(ind)==0:
      y2_all.append(temp)
    else:
      y2_all.append(y2[ind[-1]])
      temp = y2[ind[-1]]

  diff=[]
  for i in range(n):
    diff.append( np.absolute( y1_all[i] - y2_all[i]  ) )

  max_diff = np.max(diff)

  max_ind = np.argmax(diff)

  if draw:
    draw_ecdf(x1,y1,x2,y2, max_diff, max_ind)

  if max_diff > threshold:
    print(f"Max value = {max_diff} > C: {threshold}, We reject H0")
  else:
    print(f"Max value = {max_diff} <= C: {threshold}, We reject H0")
```

In [63]:
```
ks_2_sample_test(data['nd_confirmed_per_day'].values, data['ne_confirm
```

```
598
```



```
Max value = 0.7500000000000013 > C: 0.05, We reject H0
```

Result of 2 sample KS test for last 3 months of 2020 for ND state and NE state cases

Null hypothesis (H0):

Distribution of ND state cases equals distribution of NE state cases

Alternate hypothesis(H1):

Distribution of ND state cases not equals to distribution of NE state cases

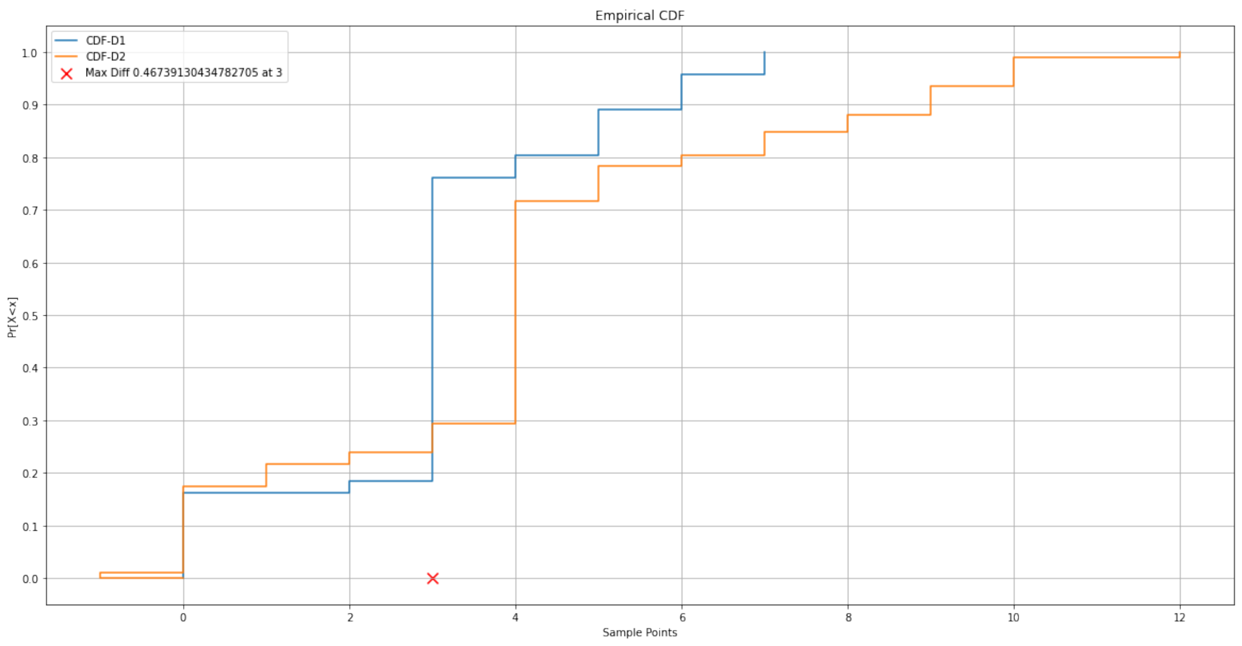
Procedure :

We have taken c = 0.05 as given in documentation and calculated the maximum difference of the CDF of the distributions at all the points.

Result:

As the KS test value is 0.75 which is greater than 0.05 we are rejecting the NULL hypothesis.

In [27]:
```
ks_2_sample_test(data['nd_deaths_per_day'].values, data['ne_deaths_per
```



```
Max value = 0.46739130434782705 > C: 0.05, We reject H0
```

Result of 2 sample KS test for last 3 months of 2020 for ND state and NE state deaths

Null hypothesis (H0):

Distribution of ND state deaths equals distribution of NE state deaths

Alternate hypothesis(H1):

Distribution of ND state deaths not equals to distribution of NE state deaths

Procedure :

We have taken the c = 0.05 as given in documentation and calculated the maximum difference of the CDF of the distributions at all the points.

Result:

As the KS test value is 0.467 which is greater than 0.05 we are rejecting the NULL hypothesis.

In [60]:
```
def permutation_test(X, Y, n=1000, threshold=0.05):
  T_obs = abs(np.mean(X) - np.mean(Y))
  xy = np.append(X,Y)
  p_value = 0.0
  T = []
  for i in range(n):
    permutation = np.random.permutation(xy)
    X1 = permutation[:len(X)]
    Y1 = permutation[len(X):]
    Ti = abs(np.mean(X1) - np.mean(Y1))
    T.append(Ti)

  T = np.array(T)
  p_value = np.sum(T>T_obs)/len(T)
  print("The p-value is: ", p_value)
  if (p_value <= threshold):
    print("p-value less than or equal to the threshold ==> Reject the
  else:
    print("p-value greater than threshold ==> Accept the Null Hypothes
```

In [61]:
```
x1 = data['nd_confirmed_per_day'].values
x2 = data['ne_confirmed_per_day'].values

y1=data['nd_deaths_per_day'].values
y2=data['ne_deaths_per_day'].values

permutation_test(x1,x2)
permutation_test(y1,y2)
```

```
The p-value is:  0.0
p-value less than or equal to the threshold ==> Reject the Null Hypot
hesis
The p-value is:  0.003
p-value less than or equal to the threshold ==> Reject the Null Hypot
hesis
```

Result of Permutation test for last 3 months of 2020 for ND state and NE state cases

Null hypothesis (H0):

Distribution of ND state cases equals distribution of NE state cases

Alternate hypothesis(H1):

Distribution of ND state cases not equals distribution of NE state cases

Result:

As the Permutation test value is 0.0 which is less than 0.05 we are rejecting the NULL hypothesis.

Result of Permutation test for last 3 months of 2020 for ND state and NE state deaths

Null hypothesis (H0):

Distribution of ND state deaths equals distribution of NE state deaths

Alternate hypothesis(H1):

Distribution of ND state deaths not equals distribution of NE state deaths

Result:

As the Permutation test value is 0.003 which is less than 0.05 we are rejecting the NULL hypothesis.

In [2]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [3]:

```
%cd gdrive/MyDrive/probstat
```

```
/content/gdrive/MyDrive/probstat
```

In [ ]:

```
%pwd
```

Out[25]:

```
'/content/gdrive/My Drive/probstat'
```

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gamma
```

In [20]:

```
def plot_gamma_distribution(posteriors):
  weeks=[]
  for i in range(5,9):
      weeks.append("week"+str(i))
  for posterior in posteriors:
    alpha=posterior[0]
    beta=posterior[1]
    x = np.linspace(gamma.ppf(0.01, alpha),gamma.ppf(0.99, alpha), 100)
    y1 = gamma.pdf(x, a=alpha)
    plt.plot(x, y1)
  plt.legend(weeks)
  plt.xlabel("x")
  plt.ylabel("pdf")
  plt.title("Posterior distributions")
  plt.show()
```

Plotting the gamma distributions of posterior

In [15]:

```
def get_new_params(prior_alpha,prior_beta,x):
  #we get the sum because of the sum of exponents of exponential
  posterior_alpha=prior_alpha+1+np.sum(x)
  # we get the sum because of the sum of exponents of lambda which is the size
  posterior_beta=prior_beta+x.size
  return posterior_alpha,posterior_beta
```

Getting the alpha and beta of the new posterior from the prior data and sample data.

In [13]:

```
def get_posteriors_for_distribution(x):
  prior_alpha=0
  prior_beta=1/np.mean(x[0:28])
  posteriors=[]
  #we can write exponential distribution as a gamma distribution with alpha=0 and beta=1/la
  for i in range(4):
    #we get the posterior distribution by adding the alpha and beta from 2 distributions
    posterior_alpha,posterior_beta=get_new_params(prior_alpha,prior_beta,x[28+(i*7):28+((i+
    posterior=[posterior_alpha,posterior_beta]
    posteriors.append(posterior)
    prior_alpha,prior_beta=posterior_alpha,posterior_beta
  return posteriors
```

In [18]:

```
def get_MAP(posteriors):
    MAPS=[]
    #differentiating the gamma w.r.t x gives (alpha-1)/beta
    for posterior in posteriors:
        alpha = posterior[0]
        beta = posterior[1]
        map=(alpha-1)/beta
        MAPS.append(map)
    return MAPS
```

Calculate the MAPs for the posterior distribution which is a gamma distribution. MAP=(alpha-1)/beta

In [10]:

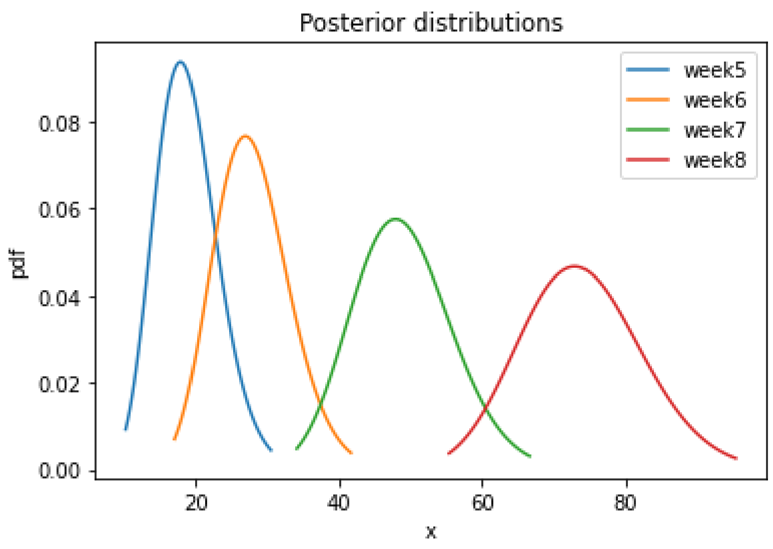
```
def q1_d():
    #get data from new file
    dataframe = pd.read_csv('../15_updated.csv')
    #get daily stats and conver to numpy
    ne_deaths = dataframe['ne_deaths_per_day'].to_numpy()
    nd_deaths = dataframe['nd_deaths_per_day'].to_numpy()

    #add the total deaths from both states
    total_deaths = ne_deaths[131:187] + nd_deaths[131:187]

    #get posterior gamma distributions for week 5,6,7,8
    posteriors = get_posteriors_for_distribution(total_deaths)
    print("posteriors", posteriors)
    #get the MAPs by finding the MLE of the gamma posterior
    maps=get_MAP(posteriors)
    print("MAPS", maps)
    #plot the posterior gamma distributions
    plot_gamma_distribution(posteriors)
```

Approach

- The exponential prior can also be written as a gamma distribution with alpha=0 and beta=mean of exponential dist=1/lambda_MME
- The exponential prior and MLE of the poission gives a gamma distribution with alpha and beta with alpha-1 being the coefficient of lambda and beta being the coefficient of the exponent.
- Taking the posterior of the previous week as the prior of the current week we get the a new gamma distribution in each

```
In [21]:
q1_d()
```

```
posteriors [[19, 7.261682242990654], [28, 14.261682242990654], [49, 21.26168
2242990652], [74, 28.261682242990652]]
MAPS [2.4787644787644787, 1.8931847968545217, 2.2575824175824177, 2.58300264
55026456]
```


Posterior distributions
week5
week6
week7
week8
pdf
x